EDB
Postgres for the AI Generation

Optimizing for Access Patterns with Table Access Methods

Tom Kincaid
SVP Database Servers and Tools
Tom.Kincaid@enterprisedb.com

Who am I -Tom Kincaid

- Tom Kincaid
- SVP for database server development for EDB
    - Manage the release of many products and our contributions to Postgres
- 15 years working with Postgres
- Run the Boston Postgres Users Group
- 25 with databases (prior to Postgres Spent a lot of time working with OODBMS).
- 40 years of product software development and support

Agenda

- Why this topic?
- The typical Postgres Database Tuning steps
- This missed opportunity: Table A Ms and Index A Ms
- Talk about what Table A Ms and Index A Ms are
- The state of this opportunity
- Thoughts on why we are where we are
- What I- hope the future looks like

Why this topic?

- Postgres is doing great
- There are **always** areas to improve
- We are missing a huge opportunity in the area of Table A M s
- Story is not non-existent but …. It could be so much better
- Hoping to inspire PGDG to do more
- Hoping to inspire users to use demand more and to use more

How does one optimize a Postgres Database?

1. Optimize the operating system for the server
2. Optimize the postgres configuration and tune it for the target deployment

Typically happens once

3. Monitor for table bloat and tune your vacuum parameters
4. Evaluate if any queries are spilling to disk and tune memory settings
5. Ensure your statistics are up to date.
6. Determine if any of your queries can benefit from indexes

Happens Regularly (Mostly by Humans, Likely will be agent n)

Missed opportunity here for access pattern optimization

7. If steps 1-6 fail, evaluate rewriting you queries

Hopefully never happens

The perpetual tuning cycle that I want

1. Optimize the operating system for the server
2. Optimize the postgres configuration and tune it for the target deployment
3. Monitor for table bloat and tune your vacuum parameters
4. Evaluate if any queries are spilling to disk and tune memory settings
5. Ensure your statistics are up to date.
6. Determine if any of your queries can benefit from indexes
7. **Optimize your tables based on observed access patterns (using Table Access Methods)**

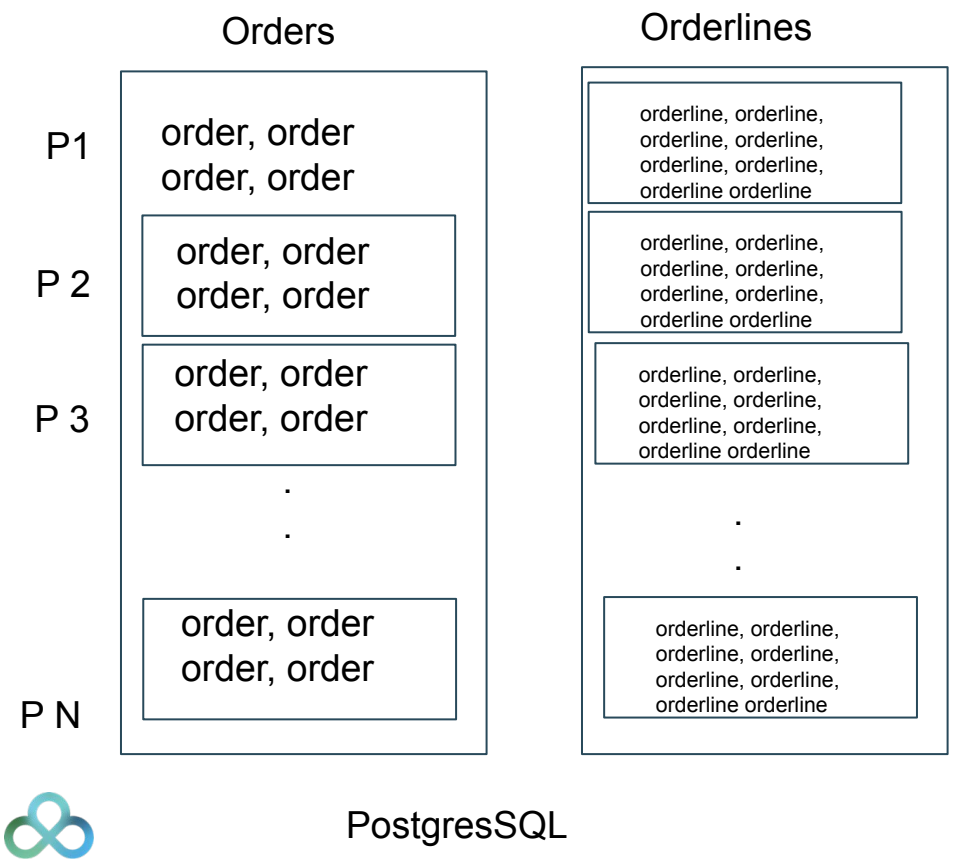
Happens Regularly (By Humans and Agents)

8. If steps 1-6 fail, evaluate rewriting you queries

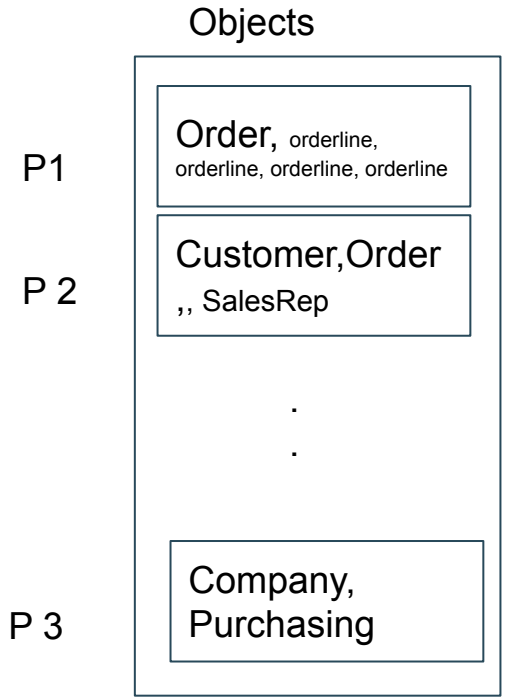
Why this topic is important to me

- My first database job was working on an Object Database
- I was the world's greatest Object Database Optimizer
- Very tight coupling with the application
- Were usually dealing with application programmers
    - You could see their eyes light up
- Hoping this presentation can inspire developers and dbas to take next steps

Object Database store things different than Postgres



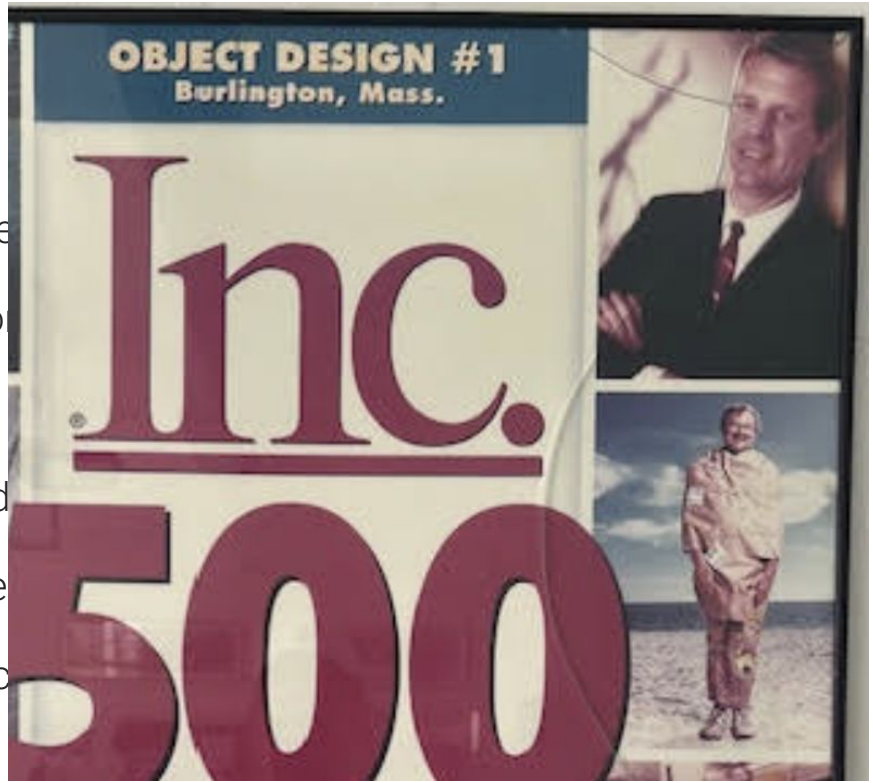
PostgresSQL



My Object Database

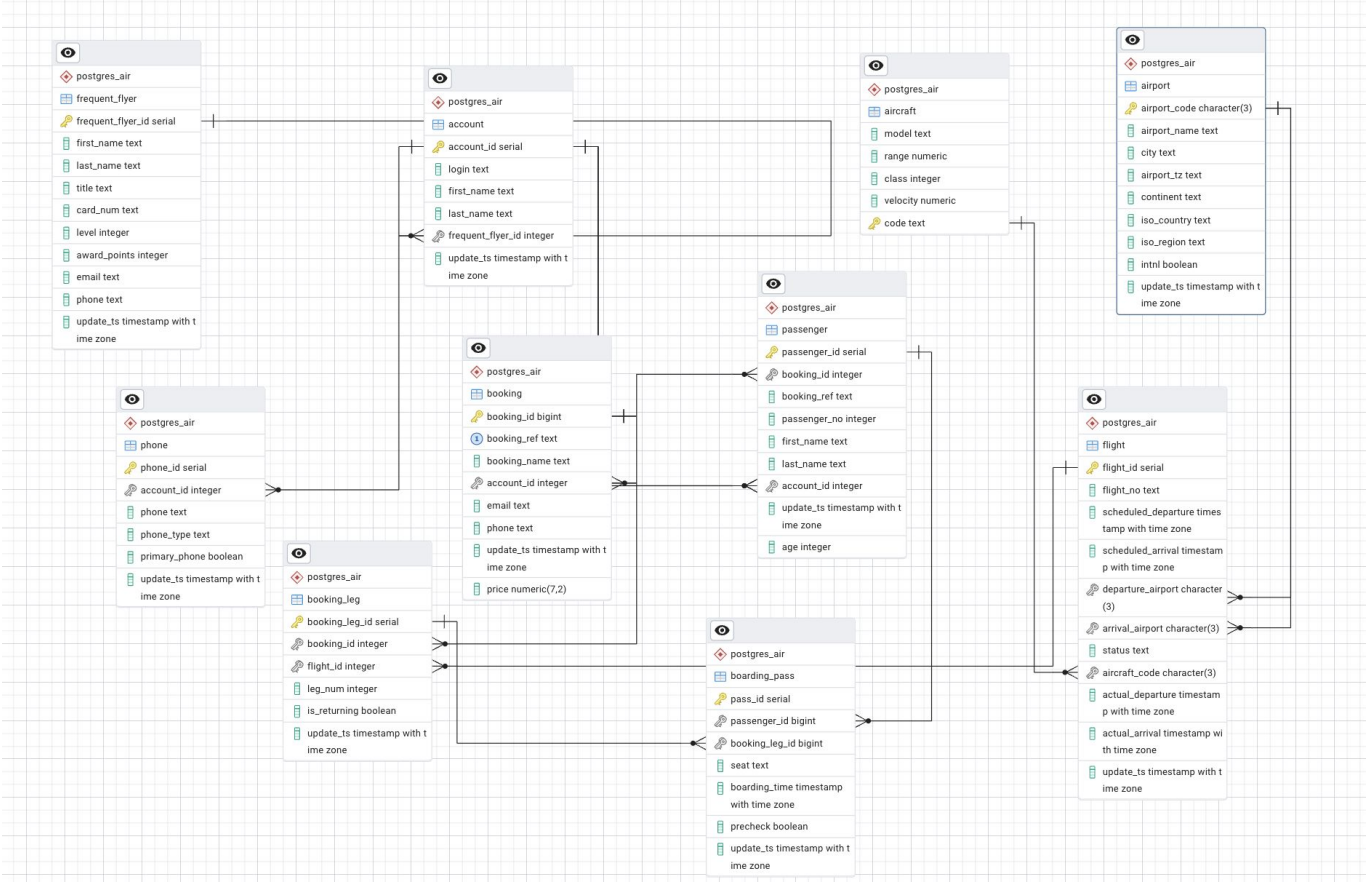
How this went down

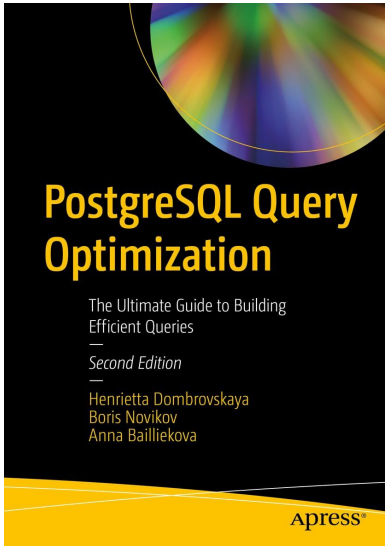
1) Awesome for particular use case / access patte
2) Application programmers loved it – Had real co
3) DBAs hated it
4) When the application changed things became d
5) As new access patterns appeared performance
6) Tooling and different programming languages c



BUT I LOVED IT !!!
AND I THINK WE CAN MAKE FOLKS EYES Light Up again with Table A Ms.

Introduction to Postgres Air





- A sample database for trying out new tricks with Postgres
- https://github.com/hettie-d/postgres_air

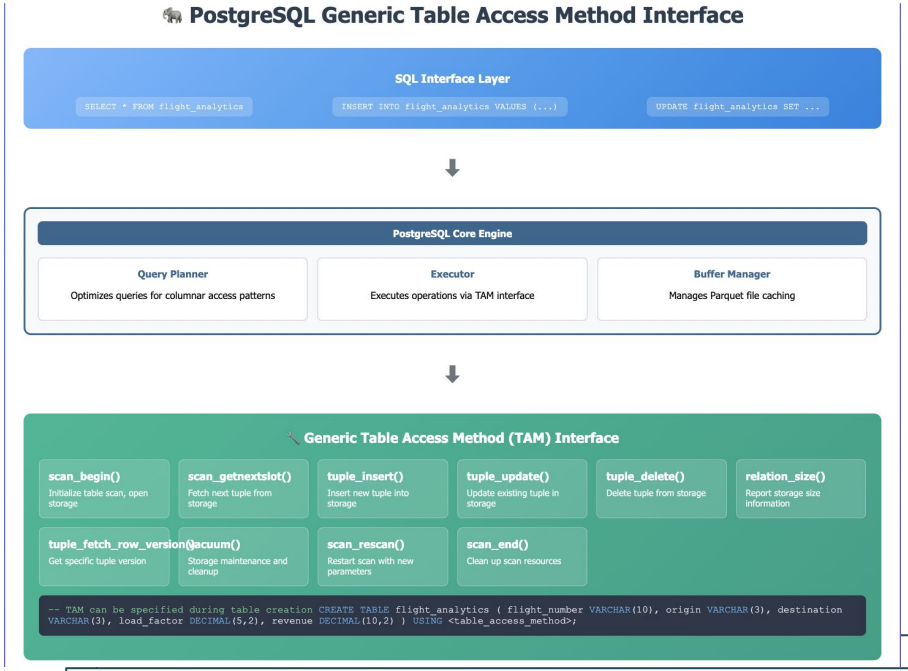
What is a Table Access Method (Table A M) ?

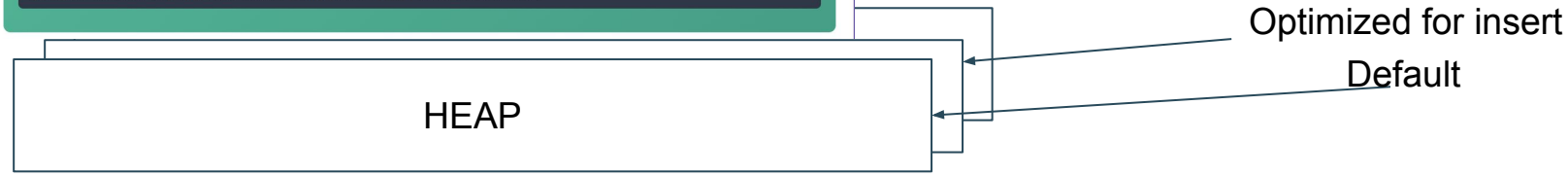
- A storage interface Postgres uses for Tables
- Abstracts a Table's storage format and storage operation from the rest of the systems
- Every table created table created in Postgres (after version 12) is interfaced via a Table A M
- The default TABLE A M is called *heap*
- Presently Postgres only distributes one Table A M (heap)

```
CREATE TABLE postgres_air.boarding_pass (
   pass_id integer DEFAULT
   passenger_id bigint,
   booking_leg_id bigint,
   seat text,
   boarding_time timestamp with time zone,
   precheck boolean,
   update_ts timestamp with time zone
);
```

```
CREATE TABLE postgres_air.boarding_pass (
   pass_id integer DEFAULT
   passenger_id bigint,
   booking_leg_id bigint,
   seat text,
   boarding_time timestamp with time zone,
   precheck boolean,
   update_ts timestamp with time zone
) using heap;
```

Tuning cycle I want includes selecting appropriate Table A MA





What if we know how table is accessed?

```
CREATE TABLE postgres_air.boarding_pass (
   pass_id integer DEFAULT
   passenger_id bigint,
   booking_leg_id bigint,
   seat text,
   boarding_time timestamp with time zone,
   precheck boolean,
   update_ts timestamp with time zone
);
```

```
CREATE TABLE postgres_air.boarding_pass (
   pass_id integer DEFAULT
   passenger_id bigint,
   booking_leg_id bigint,
   seat text,
   boarding_time timestamp with time zone,
   precheck boolean,
   update_ts timestamp with time zone
) using special_insert_only_tam;
```

We learn that the boarding_pass table is insert only and select only. How we learn is something I will cover a bit later. Perhaps even demo.

Can change a Table's A M at any time

```
CREATE EXTENSION special_insert_only_tam;
ALTER TABLE postgres_air.boarding_pass SET ACCESS METHOD special_insert_only_tam;
```

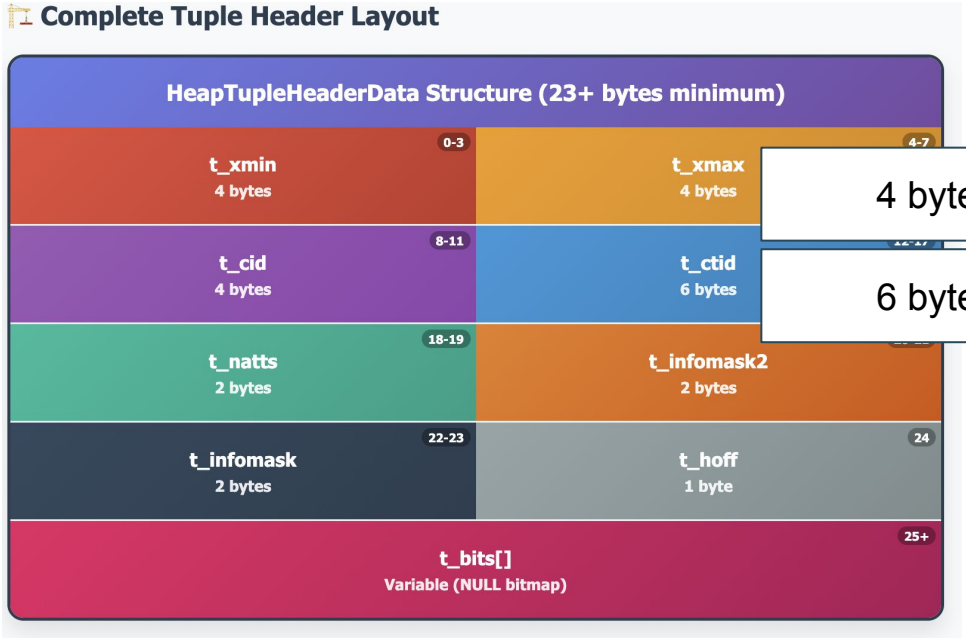
- Will take a full table lock
- Will rewrite table
- Indexes will be re-created
- It will take time but will require not application changes
- You can revert

```
ALTER TABLE postgres_air.boarding_pass SET ACCESS METHOD heap;
```

Can save space if table won't have updates or deletes

HEAP PAGE

HEAP TUPLE HEADER



4 bytes saved

6 bytes saved

I can remove at least 10 bytes from tuple header. In many cases more. Can go as low as 2 bytes in some scenarios.

Can compress against same value in tuple header

t_xmin	REST OF THE TUPLE
46	
46	
46	
46	
46	

- Can compress all rows against a top level row.

Tuples data can be compressed to the previous tuple

```
CREATE TABLE postgres_air.boarding_pass (
   pass_id integer DEFAULT
   passenger_id bigint,
   booking_leg_id bigint,
   seat text,
   boarding_time timestamp with time zone,
   precheck boolean,
   update_ts timestamp with time zone
);
```

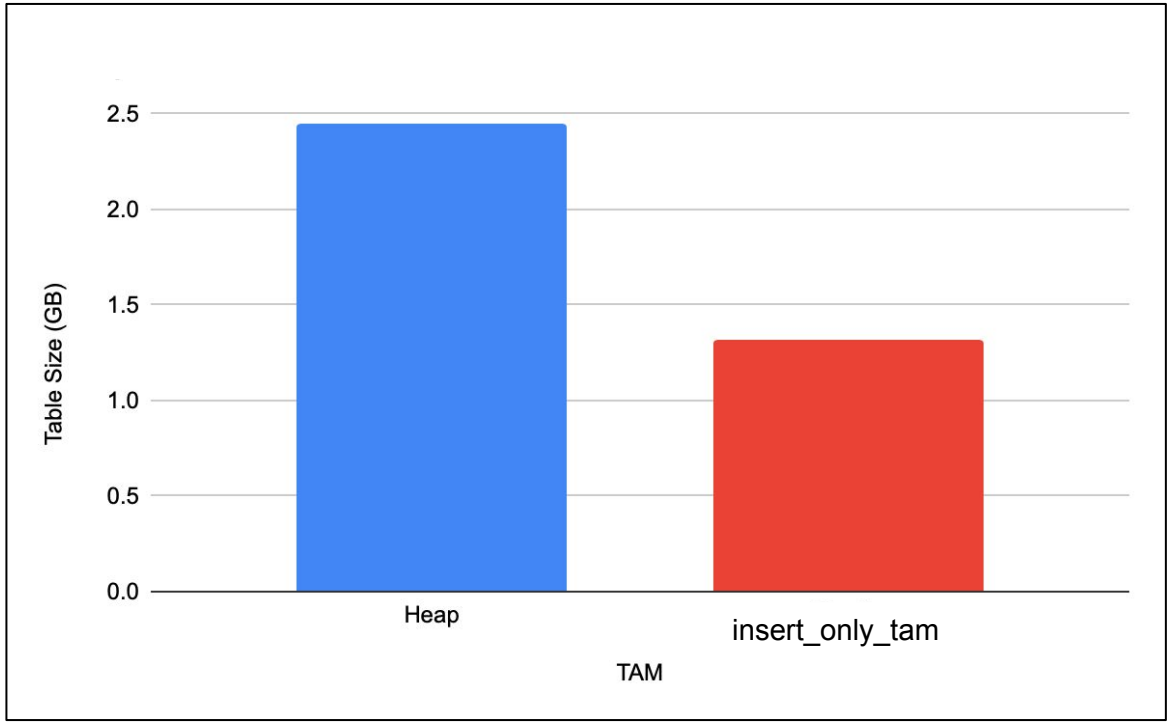
8 bytes. Number of microseconds since January 1, 2000 at 00:00:00 UTC.

Since each row likely to differ from the previous row by a small number of bytes or even bits.

You significantly reduce table size via compression against previous tuple.

How this can play out in practice (storage)

- ½ much storage needed for backups
- Cloud Storage costs reduced
- Cache hit ratio can be far better



How this can play out in practice (performance)

```
CREATE OR REPLACE PROCEDURE batch_select()
LANGUAGE plpgsql
AS $$
DECLARE
   i INT;
   passid INT;
BEGIN
   FOR i IN 1..1000 LOOP
       passid := floor(random() * 65201231 + 1);
       PERFORM passenger_id, boarding_time
       FROM postgres_air.boarding_pass
       WHERE pass_id = passid;
   END LOOP;
END;
```

- Our pgbench script calls `batch_select();`
- Use pgbench to measure performance TPS.
- One TPS is the execution of this function

How this can play out in practice (performance)

- In a particular case we see a 13x performance win

Lies, Dam Lies and Benchmarks

This was contrived benchmark but in practice is real.

A TAM for reference Tables

- By default Postgres takes a KEY SHARE lock on tables that are *referenced*
- Causes performance problems with volume inserts into tables that have foreign keys
- Examples:
  - Inserting shipping addresses into a table that has an FK to a US States Table
  - Inserting trades into a table references a stock symbol table
  - Inserting tickets into a table that references an airport table

Example of what I am referring to

```
CREATE TABLE postgres_air.flight (
   flight_id integer NOT NULL,
   flight_no text NOT NULL,
   scheduled_departure timestamp with time zone NOT
NULL,
   scheduled_arrival timestamp with time zone NOT NULL,
   departure_airport character(3) NOT NULL,
   arrival_airport character(3) NOT NULL,
   status text NOT NULL,
   aircraft_code character(3) NOT NULL,
   actual_departure timestamp with time zone,
   actual_arrival timestamp with time zone,
   update_ts timestamp with time zone
);
```

Foreign Keys to Airport Table

```
CREATE TABLE postgres_air.airport (
   airport_code character(3) NOT NULL,
   airport_name text NOT NULL,
   city text NOT NULL,
   airport_tz text NOT NULL,
   continent text,
   iso_country text,
   iso_region text,
   intnl boolean NOT NULL,
   update_ts timestamp with time zone
);
```

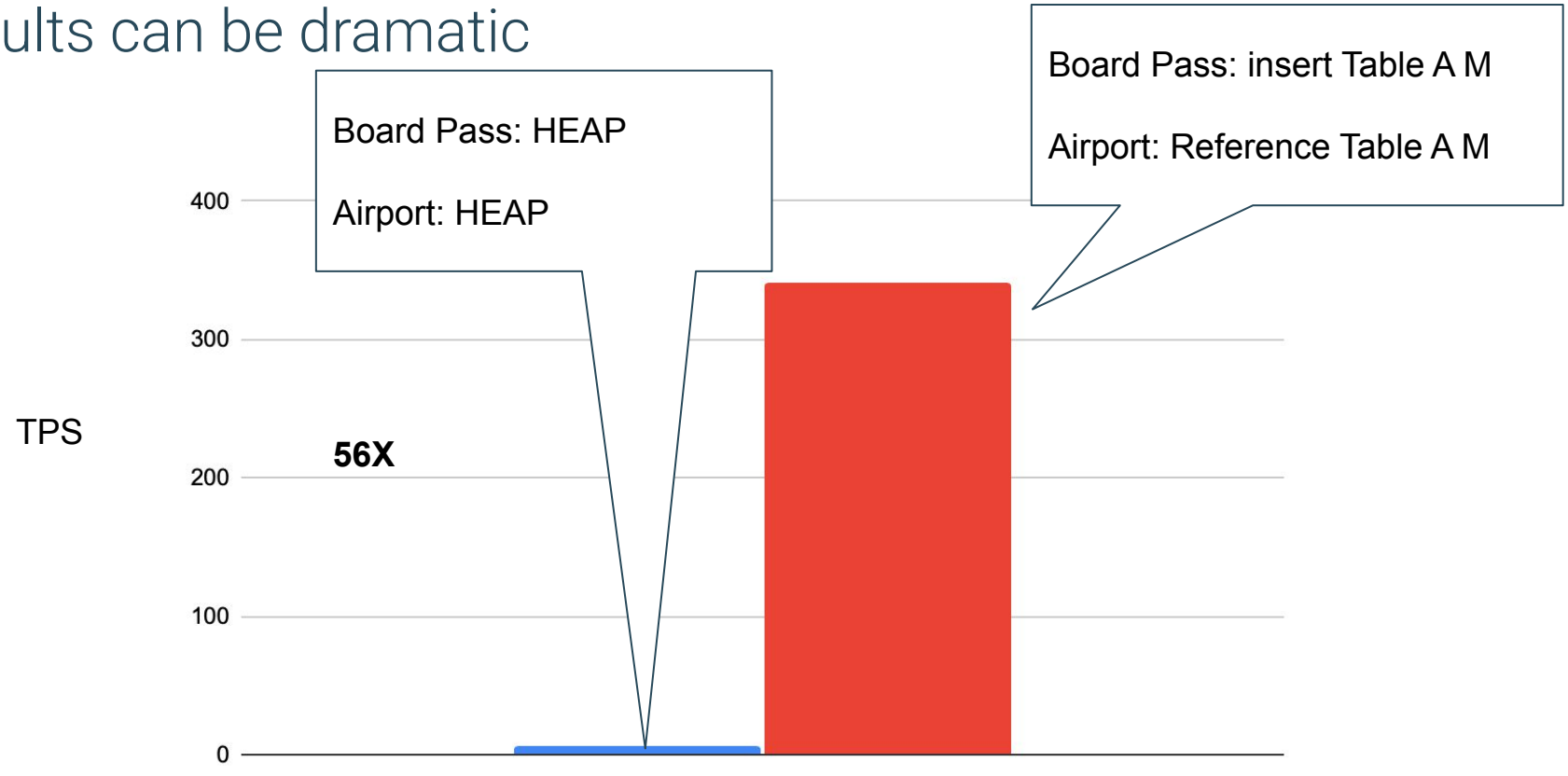
```
ALTER TABLE ONLY postgres_air.airport  ADD CONSTRAINT airport_pkey PRIMARY KEY (airport_code);

ALTER TABLE ONLY postgres_air.flight ADD CONSTRAINT arrival_airport_fk FOREIGN KEY (arrival_airport)
REFERENCES postgres_air.airport(airport_code);
```

Modify Airports to be a “referenced” Table A M

- Don’t use KEY SHARE ROW level lock on referenced row.
- Use a lighter ACCESS SHARE lock for the table
- Must use strong level locks for other operations
  - Deleting Rows
  - Updating Rows
  - Insert Rows will take longer

```
CREATE TABLE postgres_air.airport (
   airport_code character(3) NOT NULL,
   airport_name text NOT NULL,
   city text NOT NULL,
   airport_tz text NOT NULL,
   continent text,
   iso_country text,
   iso_region text,
   intnl boolean NOT NULL,
   update_ts timestamp with time zone
) using reference_data;
```

Results can be dramatic




```
INSERT INTO postgres_air.boarding_pass ( passenger_id, booking_leg_id,seat, boarding_time,precheck,update_ts) VALUES ( ....  );
200 Concurrent clients;
```

A future use case

- The thing people hate the most about Postgres is the way old rows are stored in the table
- Creates the problem of Bloat for table that have a lot of updates
- Perhaps a storage engine that stores the old tuples in an undo log
- Careful what you wish for.

```
CREATE TABLE parts_inventory (
      product_id INTEGER PRIMARY KEY REFERENCE products(id),
      quantity_available INTEGER NOT NULL,
      quantity_reserved INTEGER DEFAULT 0,
      last_updated TIMESTAMP DEFAULT now(),
      warehouse_id INTEGER REFERENCES warehouses(id)
) using frequent_update;
```

Looking the analytics use case

```
CREATE TABLE flight_performance_daily (
    flight_date DATE NOT NULL,
    flight_number VARCHAR(10) NOT NULL,
    route_id INTEGER REFERENCES routes(id),
    aircraft_id INTEGER REFERENCES aircraft(id),

    -- Passenger Metrics
    seats_sold INTEGER DEFAULT 0,
    seats_available INTEGER,
    load_factor DECIMAL(5,2),  -- Percentage occupancy
    revenue_passengers INTEGER DEFAULT 0,
    no_shows INTEGER DEFAULT 0,

    -- Revenue Metrics
    total_revenue DECIMAL(12,2) DEFAULT 0,
    average_fare DECIMAL(8,2),
    ancillary_revenue DECIMAL(10,2) DEFAULT 0,  --
Baggage, meals, etc.
```

```
-- Operational Metrics
    scheduled_departure TIMESTAMP,
    actual_departure TIMESTAMP,
    departure_delay_minutes INTEGER DEFAULT 0,
    arrival_delay_minutes INTEGER DEFAULT 0,

    -- Performance Indicators
    on_time_departure BOOLEAN DEFAULT FALSE,
    on_time_arrival BOOLEAN DEFAULT FALSE,
    flight_cancelled BOOLEAN DEFAULT FALSE,
    cancellation_reason VARCHAR(50),

    -- Fuel and Cost Metrics
    fuel_consumed_gallons INTEGER,
    fuel_cost DECIMAL(10,2),
    crew_cost DECIMAL(8,2),

    -- Updated timestamps
    last_updated TIMESTAMP DEFAULT now(),

    PRIMARY KEY (flight_date, flight_number)
) using analytics_tam;
```

Looking the analytics table

📦 Row-Based Storage

Flight	Origin	Destination	Load Factor	Revenue
AA1234	NYC	LAX	85.2%	$42,350
UA567	CHI	MIA	92.1%	$38,950
DL890	ATL	SEA	78.5%	$35,200
SW123	DEN	PHX	94.3%	$28,750

Storage: Data stored row by row
Access: Read entire rows even for single column queries

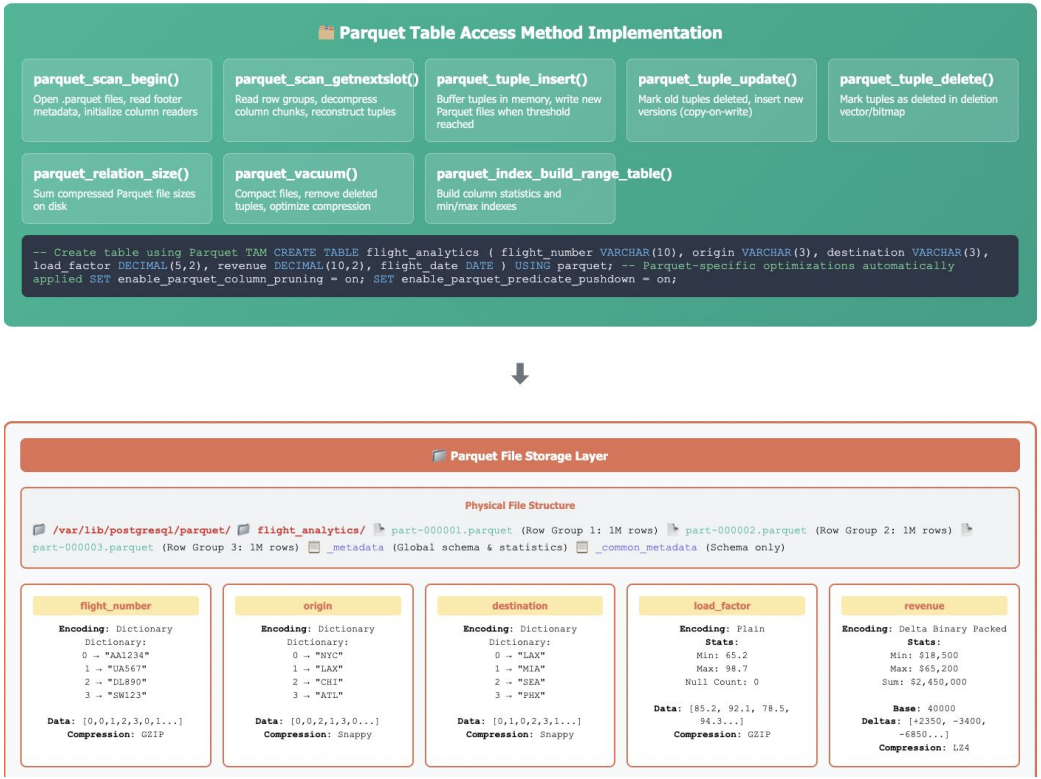
📈 Columnar Storage

Flight	Origin	Destinatio	Load Factor	Revenue
AA1234	NYC	LAX	85.2%	$42,350
UA567	CHI	MIA	92.1%	$38,950
DL890	ATL	SEA	78.5%	$35,200
SW123	DEN	PHX	94.3%	$28,750

Storage: Data stored column by column
Access: Read only needed columns for queries

More aggressive approach. Challenging to do with current TAM API.

Parquet Table Access Method



Comparing Table A Ms and FDWs

Table A M
Resides in the PG Data Directory (typically)
Administered with Postgres
Replicated to Physical Replicas
Part of backups and replication (usually)
Usually no foreign servers
Less Mature that FDWs
Development interface is challenging
Choices are still a bit limited
Need to preserve some heapness

Foreign Data Wrappers
Server will be more optimized for use case
Maturity
Interface simpler for developers
Separate Administration Interface
Transaction Consistency with Postgres
Development interface is challenging
Will likely require more machines and more skills
Will other DB vendor support you

TAM’s and the industry

- Arrived in Postgres 12 : Release October 12, 2019 six years ago

Microsoft / Citus	Columnar store TAM
OrielDB	TAM optimized for issues encountered with frequent updates and vacuum. Requires some additional patches that have not found their way to Postgres.
Hydra	Analytics Columnar Store TAM
EDB	-Insert Only Optimized Engine (Bluefin) -Reference Data Optimized Engine (refdata) -Analytics TAM offloading data to Parquet (PGAA)
Greenplum	Columnar storage engine Uses more than the Postgres TAM API.

Some general thoughts on the barriers to adoption

- Hard to develop
- TAMs are still required to have some elements of HEAP in them i.e. TIDs.
- There is only one included in standard Postgres
- There are not good recommendation tools
- DBAs are a bit concerned to use them in the wild

Some things I think will help

- Awareness – Need more success stories.
- Complete the API in Postgres
- Simplify the API
- Allow for a greater level of abstraction

Index Access Methods (IAMs)

- Unlike tables Postgres has had lots of different Index Types
- However, with AI, more need to arrive
- Postgres 18 does ads a a lot of interfaces to accommodate new IAMs

Thank you !!!

Modify Airports to be a “referenced”

- Don’t use KEY SHARE ROW level lock on referenced row.
- Use a lighter ACCESS SHARE lock
- Must use strong level locks for other operations
    - Deleting Rows
    - Updating Rows

```
CREATE TABLE postgres_air.airport (
   airport_code character(3) NOT NULL,
   airport_name text NOT NULL,
   city text NOT NULL,
   airport_tz text NOT NULL,
   continent text,
   iso_country text,
   iso_region text,
   intnl boolean NOT NULL,
   update_ts timestamp with time zone
) using reference_data;
```